**Trabalho final de Sistemas Operacionais ‘A’**

**Dênes Vargas Teixeira e Natan Luiz Berwaldt**

Universidade Federal de Santa Maria (UFSM)  
Santa Maria – RS – Brasil

dvteixeira@inf.ufsm.br nlberwaldt@inf.ufsm.br

***Abstract.*** *This report describes performance tests from an implementation of a shared thread buffer and from a simulation of a 1980s floppy drive.*

***Resumo.*** *Este relatório descreve testes de desempenho a partir de uma implementação de um buffer compartilhado entre threads e de uma simulação de uma unidade de disquete dos anos 1980.*

# **1. Objetivos**

O trabalho de análise tem dois objetivos principais, sendo o primeiro a implementação de um buffer compartilhado entre threads, a qual controle o acesso entre as threads nas regiões críticas e mantenha a ordem de inserção e remoção de acordo com a chegada dos pedidos. A segunda parte é uma implementação de uma unidade de disquete dos anos XX dupla face, usando especificações desses discos. Além disso, o programa deve realizar pedidos de leitura e escrita no disco a partir de uma imagem criada e demorar para responder o tempo aproximado que o disco real demoraria.

# **2. Metodologia**

2.1. Buffer Compartilhado

O buffer compartilhado deve possuir um tamanho fixo e poder colocar uma quantidade qualquer de dados e de tamanhos variados dependendo de sua capacidade. O buffer também deve possuir funções de inicialização, inserção de dados, remoção de dados e liberação do buffer.

O ambiente de testes será ‘n’ threads produtoras e ‘m’ consumidoras, onde n e m são variáveis, mas também serão variados os tamanhos dos dados a serem colocados e removidos do buffer.

Antes do inicio da execução das threads consumidoras deve ser inicializado o buffer que será compartilhado entre as threads, nessa inicialização é descrito o tamanho desejado do buffer a ser criado.

A execução da thread produtora, irá colocar dados no buffer compartilhado, caso o buffer possua o espaço para a inserção desejada o dado é inserido no buffer, caso contrário a thread colocada em uma fila e todas as threads produtoras são bloqueadas e as threads consumidoras são desbloqueadas. No momento que as threads são desbloqueadas elas verificam se a fila de pedidos possui alguma inserção pendente e se essa inserção é a mesma da thread produtora que conseguiu o acesso ao buffer, se esse for o caso ela exclui o pedido da fila e o executa, caso contrário ela pega o pedido da fila e coloca a sua inserção na fila, para que a ordem de chegada não seja desrespeitada.

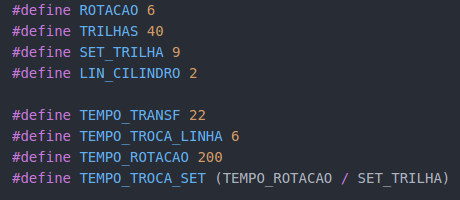
A execução da thread consumidora, é bem semelhante, enquanto ela tiver dados para remover do buffer ela continuará a executar, caso contrário ela bloqueia. Quando desbloqueada ela volta a atender os pedidos que chegarem, mas não respeita a ordem de chegada dos pedidos, o que conseguir o acesso primeiro ao buffer terá o pedido de remoção realizado, pois não foi implementado as filas na remoção do buffer.

2.2 Diskete

As informações para a implementação da simulação de um diskete foram obtidas através do livro da disciplina. As informações obtidas foram:

Velocidade de rotação;

* Número de trilhas;
* Número de setores por trilha;
* Número de trilhas por cilindro;
* Tempo de leitura de um setor;
* Tempo para trocar de uma trilha para outra adjacente;



Além é claro, de considerar o tamanho do setor como sendo de 512 Bytes. Infelizmente não foi possível implementar o tempo extra de transferência da controladora para a memória principal do sistema, considerando assim que o tempo seja o mesmo que o de transmitir um dado de dentro da função para a variável que será impressa..

Para o acesso do disco é utilizada as funções de inicialização, a qual cria o arquivo que ira simular o disco, e a função de entrelaçamento, a qual corrige o setor desejado pelo usuário para o setor real do disco e então faz o acesso ao mesmo.

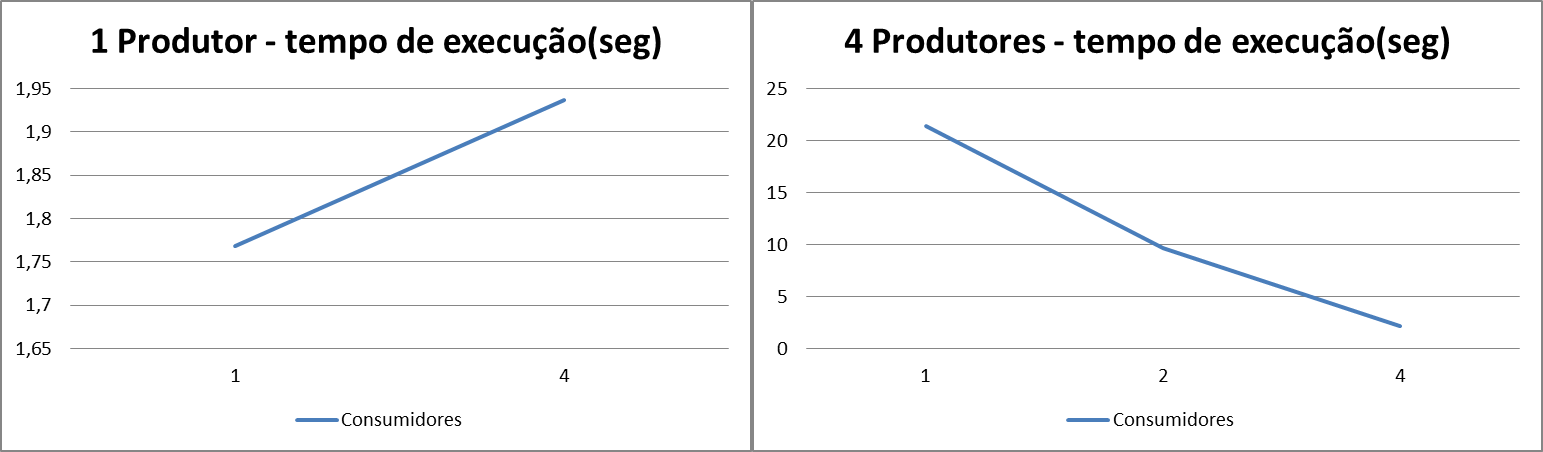
# **3. Experimentos e Resultados**

# 3.1. Buffer compartilhado

# Os testes com o buffer compartilhado foram realizados em uma máquina com processador Intel(R) Core(TM) i3-4005U com 2 núcleos e 4 processadores lógicos, com 4GB de memória RAM em um sistema operacional Windows 10 Pro x64.

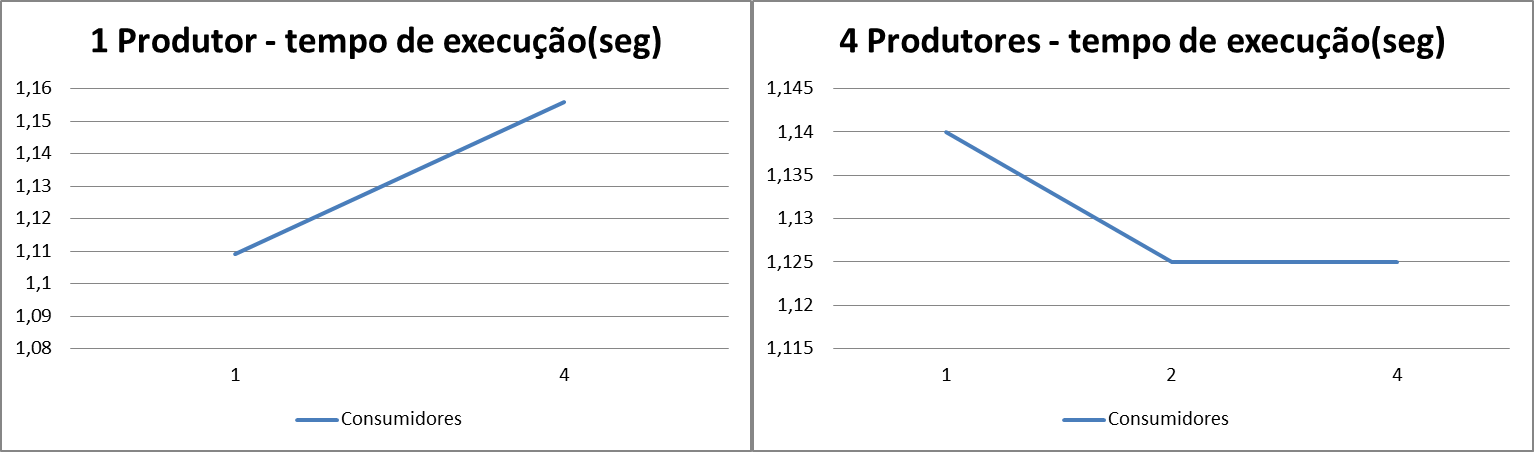
Foi usado como casos de teste os modelos de 1->1, 1->4, 4->1, 4->4 e 4->2 (consumidores -> produtores) threads, mudando o tamanho dos dados em cada caso entre 100 bytes ou 100 kB por cada inserção e remoção.

No caso de 100 bytes por pedido foi simulado em cada execução um total de 100000 acessos ao buffer, desse total cada metade é dividido entre as threads de produtores e consumidores. A partir disso mediu-se o tempo de execução nos modelos descritos anteriormente e obtiveram-se os seguintes gráficos de desempenho:



Nota-se que quanto maior a diferença no numero de threads consumidoras e produtores, o tempo aumenta drasticamente, pois já que o numero de acessos ao buffer é bem alto o numero de bloqueios das threads consequentemente também será, nisso o desempenho cai pois a cada bloqueio de threads tem um custo associado em colocar na fila de ordem de chegada e execução.

No segundo teste em que usamos 100kB por pedido, foi simulado em cada execução um total de 1000 acessos ao buffer, desse total cada metade é dividido entre as threads de produtores e consumidoras. A partir disso mediu-se o tempo de execução nos modelos descritos anteriormente e obtiveram-se os seguintes gráficos de desempenho:

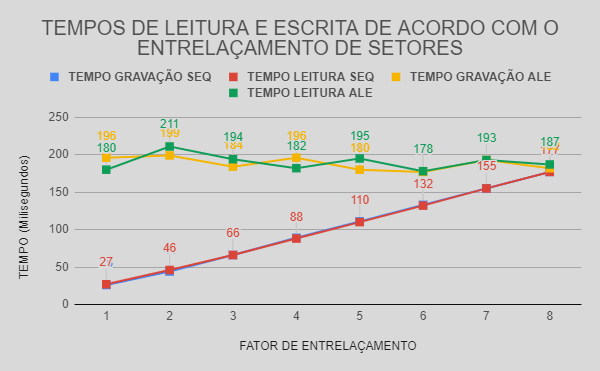


Com os resultados obtidos, pode-se notar que para dados maiores o tempo de execução ficar estável, já que a maior parte da execução se dá dentro da região critica, movendo os dados para dentro e fora do buffer o desempenho não é muito grande, porém quando se tem um número igual de threads o desempenho ainda é um pouco superior devido a gerar menos bloqueios de acesso.

3.2 Desempenho do Diskete

Os testes foram efetuados em um ambiente Linux Ubuntu, em uma máquina Intel Core i5 com 1.8Ghz de frequência e 6GB de memória RAM.

Os testes realizados são de uma média de 100 acessos ao disco em modo Escrita e 100 acessos ao disco em modo leitura, ambos feitos buscando endereços sequenciais e aleatórios, e com o fator de entrelaçamento do disco variando entre 1(Posição desejada = Posição física) e 8(Posição física = Posição desejada – 1). Totalizando assim (100\*2\*2\*8) = 320 acessos.



Podemos notar que o acesso aleatório tem seus resultados estabilizados em um cenário extremamente ruim, já que não se tem controle o ponto que será a próxima busca no disco. Já nos acessos sequenciais, o fator de entrelaçamento 1 demonstra melhores resultados, visto que devido ao pouco tempo que é gasto no processo de acesso, o cabeçote do disco quase sempre está parado exatamente em cima do setor que desejamos acessar.

# **4. Conclusão**

Os testes realizados com o buffer foram satisfatórios, exceto pela questão da fila de pedidos de remoção do buffer que não foi implementada, contudo os tempos do buffer foram esclarecedores e essa implementação nos permitirá posteriormente realizar a última parte do trabalho.

Já os testes com o disco demonstraram que com a configuração de tempos escolhida, o entrelaçamento mais efetivo foi o de fator 1, talvez pelo fato de não ter sido considerado o tempo de trânsito do dado da controladora para a memória. Por fim pode-se pelo menos ver que o quão mais eficiente é ter a possibilidade de fazer acessos sequenciais ao invés de aleatórios.